使用於 OpenFlow 網路架構下的網路狀態更新監錄機制
A Global State Consistent Update and Snapshot Mechanism for OpenFlow Switch Network

蔡逸祥　　曾黎明　　陳翔詠
國立中央大學資訊工程學系、資訊管理學系
howacha@dslab.csie.ncu.edu.tw, tsenglm@csie.ncu.edu.tw, wuddy88@dslab.csie.ncu.edu.tw

摘要

隨著網路技術快速發展，已有許多不同的網路架構紛紛推陳出新。其中近年來最受重視的是軟體定義網路(Software Define Network, SDN)，許多指標性的企業已紛紛投入。而 OpenFlow 則是最主流的軟體定義網路架構之一。

有別於傳統網路架構中的交換器，OpenFlow 交換器的狀態可以經由 OpenFlow 控制器動態設定，使得網路管理更便利。OpenFlow 的一大特色為其交換器具有動態更動的能力，可藉由類似學習的機制紀錄 flow entry。OpenFlow 帶來了許多對於網路實驗及管理好處，但若沒有良好的系統狀態儲存與更新機制，可能使得這些優勢無法發揮，本文將探討 OpenFlow switch 的監錄與更新同步問題。

本論文將探討幾個經典的分散式系統 Global snapshot 演算法及近年來應用相關演算法於 OpenFlow 的 GENE-VIOLIN snapshot 機制，以期分析及設計於 OpenFlow 架構下適用於備份及正確更新的 Global Snapshot 機制。

關鍵詞：軟體定義網路、OpenFlow、NetFPGA、全局快照、全局一致性更新

1. 緒論

隨著雲端運算技術的發展，為了有更彈性的雲端架構及動態資源配置，軟體定義網路的概念越來越受重視。其將控制層和資料層分離，使得資料網路更易於操作及管理，也擁有更好的發展彈性來面對不斷變化應用程式和各種網路條件且可利用控制中心管理網路。

目前最主流的軟體定義網路的實現方法首推美國史丹佛大學所推動的 OpenFlow[1][2]，其將傳輸路徑交由控制器全盤規劃，可視需求動態改變達成最佳化，需修改時也僅需要修改控制器的規則，大量節省了不必要的資源浪費。OpenFlow 的好處以及未來性廣受好評，目前已有許多網路設備與伺服器大廠如： Cisco、Juniper Networks、HP 等支持 OpenFlow，甚至 Google、Facebook、Microsoft 已是開放網路基金會(Open Networking Foundation, OFD)的成員，且 Google 的洲際骨幹網路已使用 OpenFlow 連接。

傳統網路的路由器或交換器在未被手動修改或更新的前提下，其狀態皆保持靜態。而在軟體定義網路中，可透過控制器修改交換器的狀態，交換器狀態則為可動態變化，因此我們可以因應網路中各種條件的變化而更新交換器狀態達成最佳化。但考量到大型分散式系統通常因為布建範圍廣大，無法利用通用時鐘(Common clock)達成絕對同步(Synchronization)，這也使得網路狀態更新的一致性成為了近年來熱門的議題。

2. 相關研究

2.1 OpenFlow

OpenFlow 改變傳統網路由交換器或路由器控管網路路徑與資料封包傳輸功能區分開來，OpenFlow 協定將資料路徑(Data Path)與控制路徑 (Control path)分離，資料轉發功能仍由交換器提供，但 Control path 的決策功能則轉移至以安裝 OpenFlow 控制軟體的控制器(Controller)來進行。OpenFlow Controller 與 OpenFlow 通過 OpenFlow Protocol 彼此通訊，其協定定義了一套訊息組，比如數據包(Datagram)接收和轉發、更改轉發路由表 flow table、以及取得交換器狀態等功能。此外，OpenFlow controller 能建立封包的處理規則，使各類封包能進行其該採取的動作，當 OpenFlow switch 處理未知封包時將通知 Controller，Controller 也將建立新處理規則，以因應未來此類型封包的處理。

2.2 Snapshot 演算法

GENI-VIOLIN[3](Global Environment for Network Innovation – Virtual Internetworking on Overlay Infrastructure)是由美國普渡大學與美國 DOCOMO 實驗室(Docomolabs-usa)於 2011 年發表，針對雲端資料中心(Cloud data center)所提出之即時分散式 snapshot 解決方案。GENI-VIOLIN 利用可程式化的 OpenFlow 交換器，在網路上提供檢查站(Checkpoint)服務，對終端主機而言所需要的改變操作可降低至最少。

GENI-VIOLIN 採用 Mattern[4]演算法，並如圖所示定義出三個時區，根據時間 t 來判斷虛擬機必處於三種可能狀態之一。Pre-Snapshot：虛擬機尚未開始 snapshot 運作，即 $t < S_i$；Snapshot：虛擬機已開始 snapshot 運作但尚未完成，即 $S_i \leq t < T_i$；Post-Snapshot：虛擬機已完成並結束 snapshot，即 t

≥ T_i。另外，根據虛擬機之間封包傳輸，將封包分成三種類型，第一型：封包由處於 snapshot 前或 snapshot 狀態之虛擬機送出且被處於 snapshot 前或 snapshot 狀態之虛擬機接收。傳送封包及接收封包之虛擬機狀態皆處於 snapshot 後也視為此類型。(圖中標示為①)；第二型：封包由處於 snapshot 前或 snapshot 狀態之虛擬機送出且被處於 snapshot 後狀態之虛擬機接收。(圖中標示為②)第三型：封包由處於 snapshot 後狀態之虛擬機送出且被處於 snapshot 前或 snapshot狀態之虛擬機接收。(圖中標示為③)

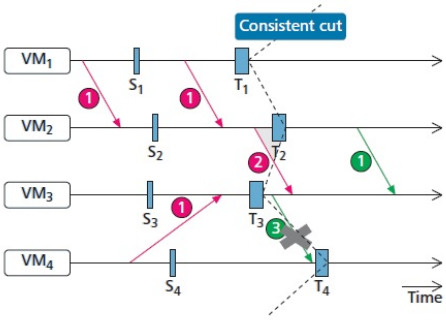

圖 1 封包及時區分類 資料來源[3]

2.3 網路版本

由於更新網路狀態無法以一個不可分割動作就完成，網路中各個交換器會陸續完成重新設定，而在開始設定至完成設定的時間差距內，則可能造成封包處理的錯誤。在此引用[5]中範例說明，如下圖所示。

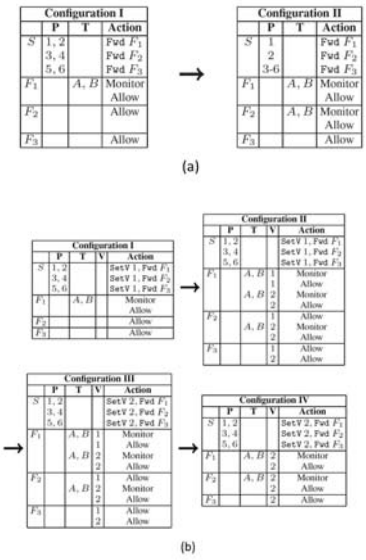

圖 2 網路設置變化 資料來源[5]

在此例中，原始網路設置如 Configuration I 所示，S 之操作為：Port 1 及 Port 2 流進之封包轉送至 F1 進行監控，而由 Port 3-6 流進之封包則分別送至 F2 及 F3，而 F2 及 F3 之管理動作為直接允許封包通過。當網路設置欲將 Port 2 流進之封包改轉送至 F2，且 F2 之管理動作也改變為監控，且 Port 3 及 Port 4 流進之封包改轉送至 F3。若直接如圖(a)所示，直接更改為 Configuration II 之狀態，那麼在操作中若我們先改變 F2 之管理動作，可能使得原本得以通過之封包被監控；若我們先改變 S 之操作，可能導致需被監控的封包被允許通過，此兩種方法皆可能發生錯誤。若採用圖(b)之方法，將封包依照版本(Version, V)而有不同的操作，可有效達成一致性更新之目的。

3. 系統設計

OpenFlow 網路可以由多個分散的交換器組成，並以一控制器為其中心管理。各交換器擁有 flow table，可儲存封包處理規則，當收到封包後選擇匹配的規則，更新計數器並執行指定操作。若在 flow table 內沒有匹配的規則，交換器將此封包的標頭(header)傳送至控制器，等待回應後再執行操作。而若各交換器無法同步更新，則可能導致封包處理錯誤，如圖 3 所示。

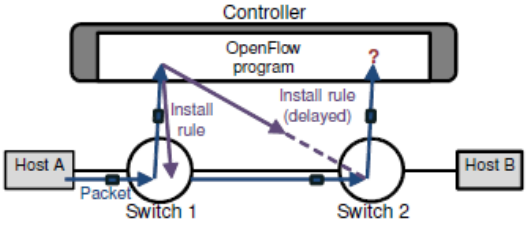

圖 3 規則更新未同步示意圖 資料來源：[6]

在此情境中，Switch 1 收到由 Host A 送出之封包，但在 flow table 內查無匹配之處理規則，將此封包標頭送至 Controller。而 Controller 安裝處理此封包之規則至 Switch 1 及 Switch 2，但因有所延遲，使得 Switch 2 收到封包時仍尚未被安裝規則，Switch 2 可能再度將此封包標頭送至 Controller 詢問，造成重複詢問而降低系統效率。此類問題易發生於佈建範圍廣大，如洲際骨幹網路系統。若封包為跨 OpenFlow 領域(domain)的傳遞，也可能發生類似問題。當封包之傳送端與接收端分屬於不同的 OpenFlow 領域時，可能因兩領域使用不同封包轉送規則時而造成封包無法正確傳送，如圖 4 所示。

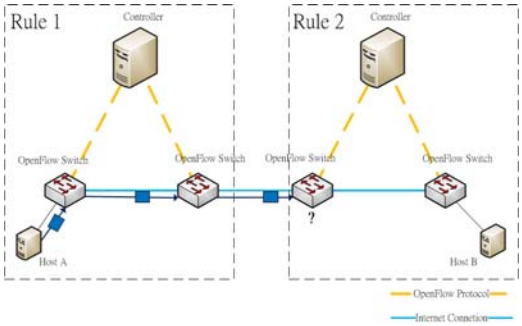

圖 4 跨領域封包傳送

3.1 系統架構

圖 5 為 OpenFlow 網路之示意圖，OpenFlow Switch 間透過網際網路連接，所有 OpenFlow Switch 皆透過 OpenFlow Protocol 接受 NOX Controller 之集中管理。本論文系統設計之假設為 OpenFlow 網路上之管理程序皆為相同之管理程序，並且具有同樣之服務層級協議(Service Level Agreement，SLA)。

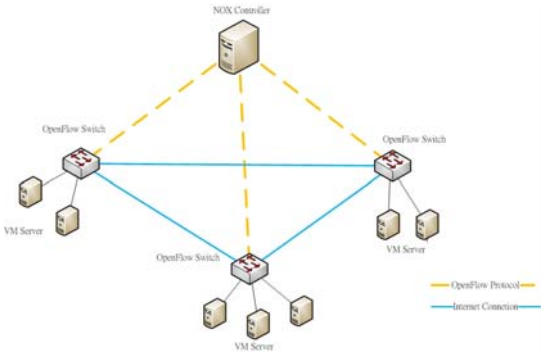

圖 5 系統架構

我們視封包轉送規則為系統之版本，當 OpenFlow switch 內之 flow table 有所更動時即代表封包轉送規則有所變化。因此當一 switch 有內部事件發生，此 OpenFlow switch 更新 flow table 後即代表為進入下一版本。我們利用 IP 封包標頭 Option 欄位來標示版本，使用 Option 內的第 9 及第 10 個位元標示版本。且系統內所有 OpenFlow switch 皆設置封包計數器(PC)，其作用為計算該版本之封包收送數量。

3.2 問題分析

本論文的 snapshot 機制及系統中各元件之運作以確保封包轉送正確性。當 OpenFlow controller 發出控制訊息至多個 OpenFlow switch 更新 flow table 時，若所有需更新之 OpenFlow switch 無法同時同步更新 flow table 的情況下，以下兩種封包類型可能會導致封包轉發錯誤：

一、 Pre-update to Post-update：當封包由尚未更新 flow table 之 OpenFlow swtich 傳出，被已更新 flow table 之 OpenFlow swtich 接收，可能導致封包傳送錯誤(如圖 6 (a)所示)。

二、 Post-update to Pre-update：當封包由已更新 flow table 之 OpenFlow switch 送出，而接收端 OpenFlow switch 之 flow table 尚未更新，可能違反因果關係(如圖 6 (b)所示)，也會造成 OpenFlow controller 負擔加重。

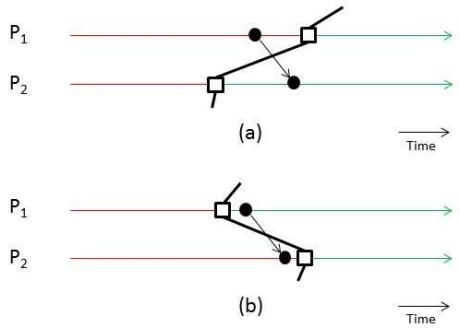

圖 6 可能造成錯誤之封包類型

3.3 演算法運作流程

3.3.1 Pre-update to Post-update

本節將探討情境：封包傳送端 OpenFlow switch 之 flow table 尚未更新，接收端 OpenFlow switch 之 flow table 已更新。在圖 7 之情境中，封包由尚未更新 flow table 的 OFS_1 送出，而封包抵達 OFS_2 前，OFS_2 已更新 flow table，將可能導致錯誤的封包轉發。

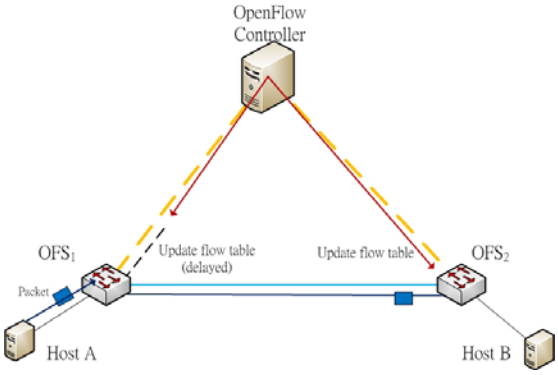

圖 7 Pre-update to Post-update 情境圖

為解決此問題，本論文機制運作之步驟如下：

初始時， OFS1 與 OFS2 之 PC 值皆為零。

1. Host A 發送封包至 OFS1。
2. OFS1 轉送封包至 OFS2，PC 值更改為 1。
3. OpenFlow controller 通知系統內所有 OpenFlow switch(即 OFS1 與 OFS2)回傳自身 PC 值，並開始統計統計系統內 PC 值，直到 PC 總和為 0 時，發送 flow table 更新訊息。
4. OFS1 與 OFS2 收到通知後即開始備份 flow table 內容，並暫停轉送由 host 發出之封包，並回傳 PC 值(此時 OFS1 回傳 1、OFS2 回傳 0)，此後若 PC 值有變動則再回傳。
5. OFS2 將封包傳送至 Host B，PC 值更改為-1 並回傳至 Controller。
6. PC 總和為 0，OpenFlow controller 發送 flow table 更新訊息。
7. OFS1 與 OFS2 更新 flow table。

更新 flow table 時依照上述執行步驟，可使各 OpenFlow switch 將需要參照目前版本之 flow table 轉發之封包皆已處理完成後才進行更新，即可解決此問題。

3.3.2 Post-update to Pre-update

本節將探討相反之情境：封包傳送端 OpenFlow switch 之 flow table 已更新，接收端 OpenFlow switch 之 flow table 尚未更新，情境示意圖如下。

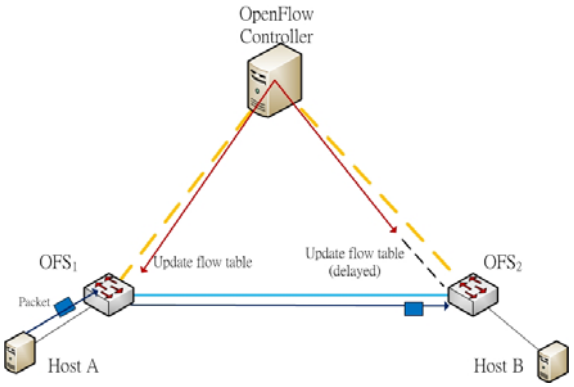

圖 8 Post-update to Pre-update 情境圖

為解決此問題，本論文機制運作之步驟如下：
初始時，OFS1 與 OFS2 之版本為 1。
1. OpenFlow controller 通知各 OpenFlow switch 更新 flow table。
2. Host A 傳送封包至 OFS1。
3. OFS1 接收來自 Host A 之封包前已先更新 flow table(OFS1 版本更新為 2)，因此使用更新後的 flow table 轉發此封包(版本為 2)至 OFS2。
4. OFS2 收到封包後檢查封包版本，因封包版本與自身版本不同，等待 flow table 更新後(版本為 2)再將封包轉送致 Host B。

3.3.3 封包處理流程

在前兩節之系統運作流程中，OpenFlow switch 收到 flow table 更新通知後也不全然即時更新，收到封包後並不全然即時處理此封包，本節將詳述 OpenFlow switch 之封包處理流程。

當 OpenFlow switch 收到封包後，首先檢查封包之版本計數器，若封包之上一站為 host，則直接執行 OpenFlow Switch 封包處理標準流程。確認封包之動作後，若動作為將此封包傳向其他 OpenFlow switch，則將封包計數器值加一後結束封包處理流程。若此封包之上一站為 OpenFlow switch，則先比較封包版本與 OpenFlow switch 版本，若封包之版本與 OpenFlow switch 之版本相同，則前往 OpenFlow 封包處理標準流程。若封包之版本與 OpenFlow switch 之版本不同，在本論文 OpenFlow switch 更新 flow table 機制下，OpenFlow switch 進入封包處理流程後並不會發生封包來自前版本之狀況，僅有封包來自後版本的可能，其相關更新機制將在下節討論。因此，OpenFlow switch 等待更新之控制訊息抵達並於更新 flow table 後，前往 OpenFlow 封包處理標準流程。確認封包之動作後，若動作為將此封包傳向 Host，則將封包計數器值減一後結束封包處理流程。

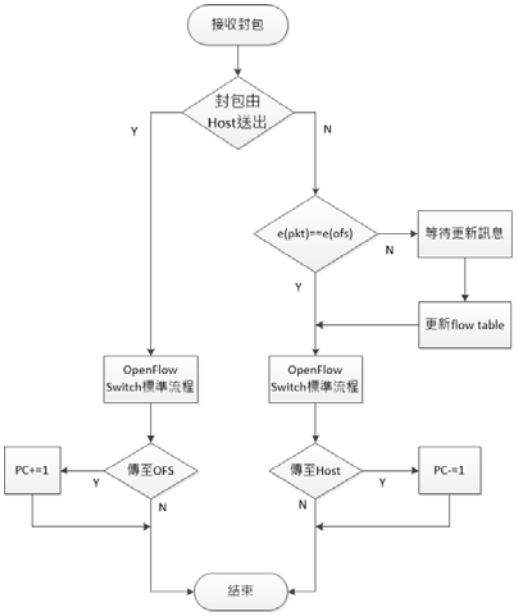

圖 9 封包處理流程圖

3.3.4 Flow table 更新流程

在上一節提到在本論文 OpenFlow switch 更新 flow table 機制下，OpenFlow switch 進入封包處理流程後並不會發生封包來自前版本之狀況，本節將詳述當封包轉發規則時 OpenFlow controller 與 OpenFlow switch 如何藉由封包計數器達成更新 flow table 時之機制，系統運作及本論文之機制步驟如下：

1. OpenFlow controller 通知與本次更新相關之 OpenFlow switch。
2. 收到更新控制訊息之 OpenFlow switch 回傳封包計數器之值，此後若封包計數器之值有所變動便再度回傳。
3. OpenFlow controller 獲得所有本次更新相關之 OpenFlow switch 之封包計數器後將其累加，若總和為 0 則通知相關之 OpenFlow switch。
4. OpenFlow switch 收到通知後更新 flow table，並將 PC 歸零。

為使此機制能適用於跨 OpenFlow 領域使用，各領域更新必須互相協調溝通，避免同時有多個控制器執行更新而造成版本不一致之問題。本論文機制採用 Leslie Lamport 所提出之 Bakery 演算法[7]實現同時僅有一控制器可執行更新之管理規則，其演算法定義小於("$\lessdot$")之關係：若(a, b) $\lessdot$ (c, d)，則(a $\lessdot$ c)或((a == c) 且 (b $\lessdot$ d))，其餘演算法如圖 10 所示。

THE BAKERY ALGORITHM: process i's code

Shared variables:

$choosing[1..n]$: boolean array
$number[1..n]$: array of type $\{0, ..., \infty\}$
Initially $\forall i : 1 \le i \le n : choosing_i = \text{false}$ and $number_i = 0$

```
1  choosing_i := true                                  /* beginning of doorway */
2  number_i := 1 + maximum({number_j | 1 ≤ j ≤ n})
3  choosing_i := false                                 /* end of doorway */
4  for j = 1 to n do
5      await choosing_j = false
6      await (number_j = 0) ∨ ([number_j, j] ≥ [number_i, i])
7  od
8  critical section
9  number_i := 0                                       /* exit code */
```

圖 10 Bakery 演算法 資料來源：[8]

4. 實驗與討論

4.1 討論

本論文機制以 OpenFlow controller 發出控制訊息更新 OpenFlow switch 之 flow table，且定義更新 flow table 為切割事件。因此，若兩 OpenFlow switch(OFS_1 及 OFS_2)分別先後收到控制訊息，可將其視為 OFS_2 送出標記至 OFS_1，如圖 11 所示。

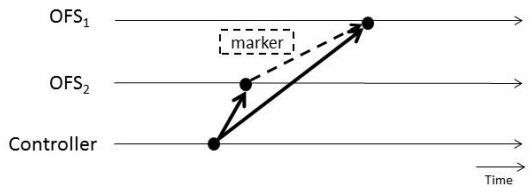

圖 11 虛擬 Marker 示意圖

非先進先出頻道會遇到的兩個問題：(1) 訊息在標記前送出，比標記晚抵達目的地；(2) 訊息在標記後送出，比標記早抵達目的地。如圖 12 所示

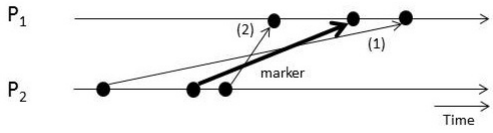

圖 12 非先進先出訊息示意圖

本論文之機制防止(1)狀況發生之操作為暫緩更新 flow table，等待所有在 flow table 更新前所送出的封包皆轉送處理完畢後再行更新，其操作結果示意圖如下。

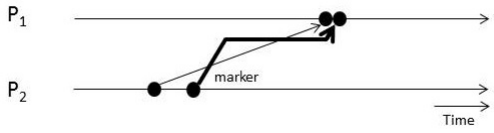

圖 14 避免 Pre-update to Post-update 示意圖

本論文之機制防止(2)狀況發生之操作如同[3]，將封包緩衝，等待接收端執行 snapshot 後再發送使其接收，其操作結果示意圖如下。

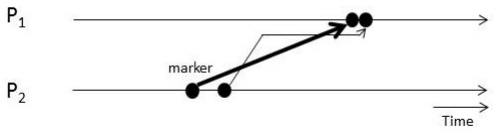

圖 14 避免 Post-update to Pre-update 示意圖

4.2 實驗

4.2.1 實驗描述

利用 NetFPGA 平台架設 OpenFlow 交換器，並交替發送不同版本之封包，測試本論文機制使 OpenFlow 交換器可將封包正確的轉送至目的地。

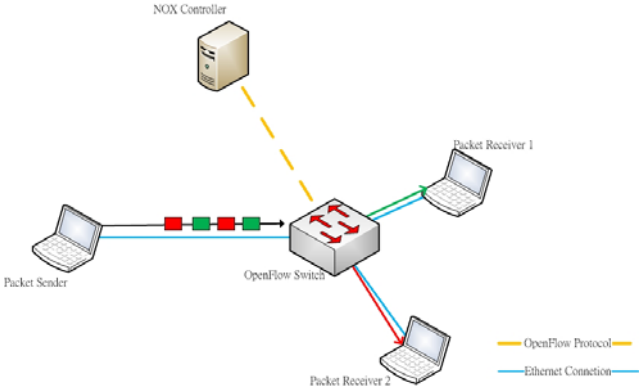

圖 15 實驗情境示意圖

實驗情境圖如圖 15，Packet Sender 將交替傳送兩種版本的封包至 OpenFlow 交換器，其中綠色封包為目前版本之封包，而此版本 OpenFlow 交換器之封包轉送規則為將封包轉送至 Packet Receiver 1，而紅色封包為下一個版本之封包，而下一版本 OpenFlow 交換器之封包轉送規則為將封包轉送至 Packet Receiver 2。

4.2.2 實驗結果

Packet Receiver 1 及 Pacekt Receiver 2 進行封包擷取，其結果圖 35，其中(a)為 Packet Receiver 1 之封包擷取結果，(b)為 Pacekt Receiver 2 封包擷取結果。

No.	Time	Source	Destination	Protoc
13	7.786862000	192.168.10.16	192.168.10.15	UDP
11	6.786934000	192.168.10.16	192.168.10.15	UDP
10	5.786984000	192.168.10.16	192.168.10.15	UDP
8	4.786992000	192.168.10.16	192.168.10.15	UDP
6	3.790593000	192.168.10.16	192.168.10.15	UDP

(a)

No.	Time	Source	Destination	Protoc
27	14.088339000	192.168.10.16	192.168.10.17	UDP
25	13.088403000	192.168.10.16	192.168.10.17	UDP
24	12.088443000	192.168.10.16	192.168.10.17	UDP
21	11.088472000	192.168.10.16	192.168.10.17	UDP
19	10.091634000	192.168.10.16	192.168.10.17	UDP

(b)

由此結果得知，第一個版本(原版本、未更新)的封

包將遵照第一個版本的封包處理規則，轉送至 Packet Receiver 1(IP 位置為 192.168.10.15)。而第二版本的封包將等待第一版本之封包全部處理完畢，OpenFlow 交換器更新封包轉發規則後，遵照規則轉送至 Packet Receiver 2(IP 位置為 192.168.10.17)。

5. 結論及未來方向

本論文針對美國史丹佛大學所提出的 OpenFlow Network 進行研究，並針對 OpenFlow switch 更新 flow table 時間不一致可能造成封包轉發錯誤提出解決方法。而經過觀察及歸納後，我們將 flow table 更新視為版本的變換，並將可能造成問題的封包類型分為兩類：

1. 封包由舊版本傳至新版本：傳送封包之 OpenFlow Switch 尚未更新 flow table，但接收封包之 OpenFlow Siwtch 在 flow table 更新後才收到封包。
2. 封包由新版本傳至舊版本：傳送封包之 OpenFlow Switch 已更新 flow table，但接收封包之 OpenFlow Siwtch 尚未更新 flow table 就收到封包。

此兩類型封包與分散式系統執行 Snapshot 時面臨的問題雷同，其中第一類型可能造成封包遺失，第二類型可能造成違反因果關係一致性。因此我們接著探討幾個著名的分散式系統 Snapshot 演算法，分析其作法及優缺點。

我們提出適用於當 OpenFlow 更新 flow table 時，仍可保持封包傳送端與接收端皆使用相同版本 flow table 之機制，保證封包可被正確轉送。最終我們經由理論證明及利用 NetFPGA 平台所架設的 OpenFlow 交換器實測，證明本論文提出的機制可有效防範當 OpenFlow Switch 更新時間不一致而造成不可預測的封包轉發情況，也避免同樣的封包在傳送路徑上因安裝規則的延遲而重複詢問 controller，提升系統執行正確性及減輕 controller 負擔。

本論文所提出的 snapshot 機制限制開始執行 snapshot 至 flow table 完成更新的這段時間內，OpenFlow 交換器將停止轉送非來自其他 OpenFlow 交換器的封包，可能造成封包傳遞的延遲，在此我們視為為了追求正確性的權宜考量。若未來能減少 OpenFlow 交換器暫停處理新封包的時間，甚至不需要暫停的機制，將可更增進系統效能。

參考文獻

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69-74, 2008.

[2] OpenFlow : OpenFlow Switch Specification Version 1.3.1September 6, 2012 https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.1.pdf

[3] X. Jiang and D. Xu, “VIOLIN: Virtual Internetworking on Overlay INfrastructure,” Technical Report CSD TR 03-027, Purdue University, 2003.

[4] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *Journal of Parallel and Distributed Computing*, vol. 18, 1993.

[5] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011, p. 7.

[6] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford, "A NICE way to test OpenFlow applications," *NSDI, Apr*, 2012.

[7] L. Lamport, "A new solution of Dijkstra's concurrent programming problem," *Communications of the ACM*, vol. 17, pp. 453-455, 1974.

[8] G. Taubenfeld, "The black-white bakery algorithm and related bounded-space, adaptive, local-spinning and FIFO algorithms," in *Distributed Computing*, ed: Springer, 2004, pp. 56-70.